

10 Class Isotope

10.1 Overview

The class Isotope defines the basic physical attributes of an isotope. These essential quantities are the number of Isotopes and their associated spin angular momentum. Functions are provided for simplified access to many isotope quantities and for disk I/O.

10.2 Available Isotope Functions

Algebraic Operators

Isotope	- Constructor	page 202
=	- Assignment	page 202
==	- Equality	page 203
!=	- Inequality	page 203

Basic Functions

qn	- Number of Isotopes (equal to Isotopes, for consistency)	page 205
HS	- Retrieve spin or Isotope Hilbert space	page 205
momentum	- Set or retrieve spin isotope type	page 206
symbol	- Retrieve spin isotope type as a string, e.g. 19F.	page 206
name	- Retrieve spin angular momentum of a spin or the system.	page 207
element	- Retrieve spin element type as a string, e.g. Carbon.	page 207
number	- Retrieve spin angular momentum as a string.	page 207
mass	- Retrieve gyromagnetic ratio of a spin.	page 208
weight	- Set or retrieve Isotope name.	page 208
mass	- Set or retrieve Isotope name.	page 208
relative_frequency	- Set or retrieve Isotope name.	page 209
gamma	- Set or retrieve Isotope name.	page 210
receptivity	- Set or retrieve Isotope name.	page 209

I/O Functions

<<	- Send Isotope to an output stream	page 212
write	- Write spin sys to disk file (as a parameter set).	page 210
read	- Read spin sys from disk file (from a parameter set).	page 211
print	- Send Isotope to output stream.	page 211

10.3 Isotope Figures

Components of Class Isotope Variables	page 213
---------------------------------------	----------

Class Isotope Functioning

page 214

10.4 Isotope Example Programs

10.5 Algebraic Operators

10.5.1 Isotope

Usage:

```
#include <Isotope.h>
void Isotope::Isotope(const Isotope& data)
void Isotope::Isotope(String symbol)
void Isotope::Isotope(Isotope &I)
```

Description:

The function *Isotope* is used to create a new isotope.

1. Isotope() - Called without arguments the function creates a default isotope. The default isotope type is a proton.
2. Isotope(String) - Called with a string, the function will create a Isotope of the size indicated. By default, all Isotopes are set to be protons (spin I = 1/2). All other Isotope parameters are left unassigned although the appropriate array space for storage of all Isotope parameters will be allocated
3. Isotope(Isotope &I) - Called with isotope, the function will make a Isotope which is a copy of the given Isotope I.

Return Value:

Isotope returns no parameters. It is used strictly to create a Isotope.

Examples:

```
#include <Isotope.h>
Isotope I;                                // A default Isotope, I, which will be a proton.
Isotope N14(String("14N"));                // An Isotope of 14N called N14.
Isotope N(N14);                           // An Isotope called N which is a duplicate of N14
```

See Also: =

10.5.2 =

Usage:

```
#include <Isotope.h>
void Isotope::operator = (const Isotope& I)
```

Description:

The unary operator = (the assignment operator) is allows for the setting of one Isotope to another Isotope. If the Isotope exists it will be overwritten by the assigned Isotope.

Return Value:

None, the function is void

Examples:

```
#include <Isotope.h>
Isotope I(String("13C"));           // Set Isotope I as a carbon 13.
Isotope I1;                         // Set a second Isotope default (1H)
I1 = I;                            // Now I1 is a carbon 13
```

See Also: Isotope, read

10.5.3 ==**Usage:**

```
#include <Isotope.h>
int Isotope::operator == (const Isotope& I) const
```

Description:

The unary operator == (the equality operator) will test two Isotopes for their equality. If the two Isotopes are identical the function returns true. If the two systems are not equal the function returns false. This function does not check the settings of the spin flags!

Return Value:

The function returns an integer which equates to TRUE or FALSE.

Example:

```
#include <Isotope.h>
Isotope AX;                      // define a NULL Isotope called AX.
Isotope A2BX(4);                 // define Isotope A2BX containing four Isotopes.
AX = A2BX;                       // set AX equal to A2BX.
if(A == A2BX)                    // see if they are in fact the same.
    cout << "The two Isotopes are identical";
```

See Also: !=

10.5.4 !=**Usage:**

```
#include <Isotope.h>
int Isotope::operator != (const Isotope& I) const
```

Description:

The unary operator != (the inequality operator) can be used to test whether two Isotopes are equal. The function returns TRUE if the two Isotopes being compared are not equal and FALSE if they are identical.

Return Value:

TRUE or FALSE.

Example:

```
#include <Isotope.h>
Isotope I1, I2;                                // Define two Isotopes called AX.
Isotope A2BX(4);                               // define Isotope A2BX containing four Isotopes.
if(AX != A2BX)                                  // see if they are not the same.
    cout << "The two Isotopes are different";
```

See Also: ==

10.6 Basic Functions

10.6.1 qn

Usage:

```
#include <Isotope.h>
double Isotope::qn () const
```

Description:

The function *qn* is used to obtain the quantum number I (spin angular momentum) carried by an Isotope. These are output in units of hbar, the default (for proton) being 0.5. When no spin is specified the function returns the sum of all spin I's.

Return Value:

A floating point number, double precision.

Example(s):

```
#include <Isotope.h>
#include <stream.h>
CH3.Isotope(3);           // define Isotope CH3 containing three Isotopes.
cout << CH3.qn (2);      // Prints I value 1.5 of spin 2 in Isotope CH3.
```

See Also: momentum, isotope

10.6.2 HS

Usage:

```
#include <Isotope.h>
int Isotope::HS ( ) const
```

Description:

The function *HS* is used to access the dimension of the spin Hilbert space associated with the individual Isotope. For a single spin, the spin Hilbert space size is related to the spin quantum number by equation equation (15-1) on page 214,

$$HS(i) = 2I_i + 1$$

where *i* is the spin label and I_i the associated spin quantum number.

Return Value:

Integer value for the dimension of the Hilbert space.

Example:

```
#include <Isotope.h>
```

```
int i;
Isotope CH3(3);           // define Isotope CH3 containing three Isotopes.
cout << CH3.HS();          // output Hilbert Space size (8 by default, all I=1/2);
```

See Also:

10.6.3 momentum

Usage:

```
#include <Isotope.h>
String Isotope::momentum () const
```

Description:

The function *momentum* is used to obtain, in string format, the spin angular momentum carried by an *Isotope*.
This function is used for formatted output.

Return Value:

The function returns a pointer to a string.

Example:

```
#include <Isotope.h>
CH3.Isotope(3);           // define Isotope CH3 containing three Isotopes.
cout << CH3.momentum (2); // Prints momentum 3/2 of spin 2 in Isotope CH3.
```

See Also: qn, isotope

10.6.4 symbol

Usage:

```
#include <Isotope.h>
const String Isotope::symbol( ) const
```

Description:

The function *symbol* is used to obtain, in string format, the isotope type. This function is typically used for formatted output.

Return Value:

A String.

Example:

```
#include <Isotope.h>
#include <stream.h>
CH3.Isotope(3);           // define Isotope CH3 containing three Isotopes.
cout << CH3.symbol(2);    // Prints isotope type 23Na of spin 2 in CH3.
```

See Also: isotope, element

10.6.5 name

Usage:

```
#include <Isotope.h>
const String& Isotope::name() const
```

Description:

name is used to specify or retrieve a Isotope name.

Return Value:

None.

Example(s):

```
#include <Isotope.h>
```

See Also:

10.6.6 element

Usage:

```
#include <Isotope.h>
const String Isotope::element() const
```

Description:

The function *element* is used to obtain the name of the element assigned to a spin via the function *isotope*. It does not alter the Isotope and will return a blank if the spin has not previously been assigned. The function is commonly used for formatted output.

Return: Pointer to a string containing element label.

Example:

```
#include <Isotope.h>
Isotope I(String("51V"));           // Set Isotope I to Vanadium 51.
cout << I.element();               // Prints V to standard output.
```

See Also: isotope, symbol.

10.6.7 number

Usage:

```
#include <Isotope.h>
int Isotope::number() const
```

Description:

The function *number* is used to obtain, in string format, the spin angular momentum carried by an *Isotope*.

This function is used for formatted output.

Return Value:

The function returns a pointer to a string.

Example:

```
#include <Isotope.h>
Isotope I(String("51V"));           // Set Isotope I to Vanadium 51.
cout << I.element();               // Prints V to standard output.
```

See Also: **qn, isotope**

10.6.8 mass

Usage:

```
#include <Isotope.h>
int Isotope::mass( ) const
```

Description:

The function *number* is used to obtain, in string format, the spin angular momentum carried by an *Isotope*.
This function is used for formatted output.

Return Value:

The function returns a pointer to a string.

Example:

```
#include <Isotope.h>
CH3.Isotope(3);                  // define Isotope CH3 containing three Isotopes.
cout << CH3.momentum (2);        // Prints momentum 3/2 of spin 2 in Isotope CH3.
```

See Also: **qn, isotope**

10.6.9 weight

Usage:

```
#include <Isotope.h>
double Isotope::weight ( ) const
```

Description:

The function *number* is used to obtain, in string format, the spin angular momentum carried by an *Isotope*.
This function is used for formatted output.

Return Value:

The function returns a pointer to a string.

Example:

```
#include <Isotope.h>
CH3.Isotope(3);           // define Isotope CH3 containing three Isotopes.
cout << CH3.momentum (2); // Prints momentum 3/2 of spin 2 in Isotope CH3.
```

See Also: qn, isotope

10.6.10 receptivity

Usage:

```
#include <Isotope.h>
double Isotope::receptivity ( ) const
```

Description:

The function *number* is used to obtain, in string format, the spin angular momentum carried by an *Isotope*.
This function is used for formatted output.

Return Value:

The function returns a pointer to a string.

Example:

```
#include <Isotope.h>
CH3.Isotope(3);           // define Isotope CH3 containing three Isotopes.
cout << CH3.momentum (2); // Prints momentum 3/2 of spin 2 in Isotope CH3.
```

See Also: qn, isotope

10.6.11 relative_frequency

Usage:

```
#include <Isotope.h>
double Isotope::relative frequency( ) const
```

Description:

The function *number* is used to obtain, in string format, the spin angular momentum carried by an *Isotope*.
This function is used for formatted output.

Return Value:

The function returns a pointer to a string.

Example:

```
#include <Isotope.h>
CH3.Isotope(3);           // define Isotope CH3 containing three Isotopes.
cout << CH3.momentum (2); // Prints momentum 3/2 of spin 2 in Isotope CH3.
```

See Also: qn, isotope

10.6.12 gamma

Usage:

```
#include <Isotope.h>
double Isotope::gamma () const
```

Description:

The function *gamma* will return a value for the gyromagnetic ratio of an *isotope*. Gamma values are used to set relative frequencies and in some relaxation computations. In GAMMA, gyromagnetic ratios are maintained in SI units, T⁻¹ sec⁻¹. For example, the returned value for ¹⁹F is 2.51719e+08 and that of a proton 2.67519e+08.

Return Value:

Double, gyromagnetic ratio of spin with units rad T⁻¹ sec⁻¹.

Example:

```
#include <Isotope.h>
CH3.Isotope(3);           // define Isotope CH3 containing three Isotopes.
cout << CH3.gamma(2);    // Prints gyromagnetic ratio spin 2, 2.67519e+08.
```

See Also: isotope.

10.7 I/O Functions

10.7.1 write

Usage:

```
#include <Isotope.h>
void Isotope::write(const String& filename);
```

Description:

The function *write* enables one to specifically write the Isotope out to a readable file. The parameters are written in the standard parameter set format and readable by the framework.

Return Value:

None.

Example:

```
#include <Isotope.h>
Isotope sys(3);           // define Isotope sys containing three Isotopes.
sys.write("test.sys");    // write information in Isotope sys to file test.sys.
```

See Also: `read`

10.7.2 `read`

Usage:

```
#include <Isotope.h>
void Isotope::read (const String& filename);
```

Description:

The function `read` enables one to read in the Isotope from an external file. The file is assumed to be written in the standard parameter set format (see the Section Spin Syst Parameter Files later in this Chapter) utilizing the standard parameter names associated with a Isotope. Typically, the file being read was generated from a previous simulation through the use of the analogous Isotopetem function `write`. Alternatively, one can use an editor to construct a parameter set file for the Isotope.

Return Value:

None.

Example(s):

```
#include <Isotope.h>
Isotope sys;                                // define an empty Isotope.
sys.read("test.sys");                         // read in the Isotope from file test.sys.
```

See Also: `write`

10.7.3 `print`

Usage:

```
#include <Isotope.h>
ostream& Isotope::print(ostream&) const
```

Description:

The function `print` is used to print out all the current information stored in a Isotope object. This includes the number of Isotopes and spin isotope types.

Return Value:

None.

Example(s):

```
#include <Isotope.h>
CH3.Isotope(3);                            // define Isotope CH3 containing three Isotopes.
CH3.print();                               // print out information in Isotope CH3.
```

10.7.4 <<

Usage:

```
#include <Isotope.h>
ostream& operator<< (ostream&, Isotope&)
```

Description:

The operator << is used for standard output of a Isotope. This includes the number of Isotopes and spin isotope types.

Return Value:

None.

Example(s):

```
#include <Isotope.h>
CH3.Isotope(3);           // define Isotope CH3 containing three Isotopes.
cout << CH3;              // write the Isotope CH3 to standard output.
```

See Also: **write, read, print**

10.8 Description

An Isotope is an atom which has specific properties which. In GAMMA, class Isotope is Two fundamental quantities of a Isotope are the **number of Isotopes** it contains and the intrinsic **spin angular momentum** (I) of each spin. These factors set the dimension of the associated spin **Hilbert space**.

Components of Class Isotope Variables

Isotope Definition

1. A Pix Into an Isotope List
2. A Static Isotope List

Figure 16-4 Contents of each Isotope variable. The Isotope list is shared by all Isotope variables. The Pix points to an item in the Isotope list, the item being type IsotopeData.

Thus, each variable of Class Isotope is just a pointer to an entry in a list of Isotopes. All variables of type Isotopes know of and share the same Isotope list.

When a variable of type Isotope is declared in a GAMMA program, class Isotope automatically checks the static Isotope list to see if it exists. If it is found empty, an attempt is made to fill the list from an external file called “**Isotope.list**”. Thus it is important that GAMMA knows where to find this file if one is to use Isotopes in a program. The search path which is taken is the following:

1. library_path:
2. GAMMA_PATH
3. The local directory -

Once this list has been successfully accessed, all Isotopes in the program will access it - insuring that the list is generated and stored only once in any program. In all other Isotope functions, the Isotope uses its Pix into the list to retrieve information. The Pix points to the proper item in the list,

Class Isotope Functioning

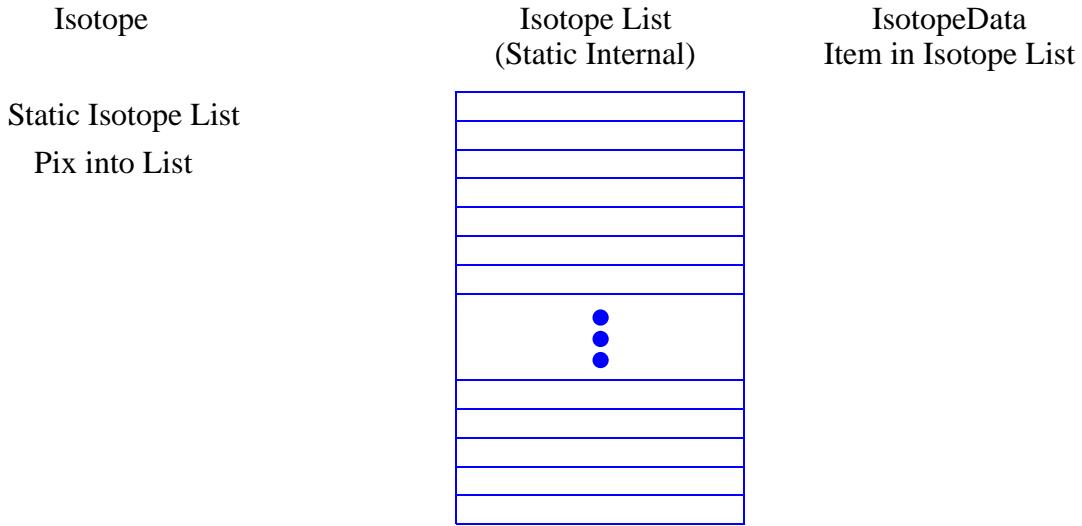


Figure 16-5 Each Isotope has a Pix which points to an item in an interal list shared by all variables of type Isotope. Each list item is of type IsotopeData. In turn Isotope data contains the information specific to a particular isotope and functions which access this information. Many Isotopes may contain the same Pix, i.e. they may be the same isotope type.

which is uniform formMathematically the Hilbert space for a single spin i is given by

$$HS(i) = 2I_i + 1 \quad (15-1)$$

where i is the spin label and I_i the associated spin quantum number.

10.9 Isotope Example Programs

This section contains example programs which were used to test Isotope functions and/or generate some of the documentation. A brief explanation is included with each program and these are referred to in the Chapter text. They should be included in the GAMMA program package in the /gamma/tests/Isotope subdirectory.

CSA_SpinT.cc

Generate Components

```
/* qStates.cc *****/
```

10.10 List of Available Isotopes

Element	I	Element	I
1H - Hydrogen	1/2	2H - Deuterium	1
3H - Tritium	1/2	3He - Helium	1/2
6Li - Lithium	1	7Li - Lithium	3/2
9Be - Beryllium	3/2	10B - Boron	3
11B - Boron	3/2	13C - Carbon	1/2
14N - Nitrogen	1	15N - Nitrogen	1/2
17O - Oxygen	5/2	19F - Fluorine	1/2
21Ne - Neon	3/2	23Na - Sodium	3/2
25Mg - Magnesium	5/2	27Al - Aluminum	5/2
29Si - Silicon	1/2	31P - Phosphorus	1/2
33S - Sulfur	3/2	35Cl - Chlorine	3/2
37Cl - Chlorine	3/2	39K - Potassium	3/2
41K - Potassium	3/2	43Ca - Calcium	7/2
45Sc - Scandium	7/2	47Ti - Titanium	5/2
49Ti - Titanium	7/2	50V - Vanadium	6
51V - Vanadium	7/2	53Cr - Chromium	3/2
55Mn - Manganese	5/2	57Fe - Iron	1/2
59Co - Cobalt	7/2	61Ni - Nickel	3/2
63Cu - Copper	3/2	65Cu - Copper	3/2
67Zn - Zinc	5/2	69Ga - Gallium	3/2
71Ga - Gallium	3/2	73Ge - Germanium	9/2
75As - Arsenic	3/2	77Se - Selenium	1/2
79Br - Bromine	3/2	81Br - Bromine	3/2
85Rb - Rubidium	5/2	87Rb - Rubidium	3/2
87Sr - Strontium	9/2	89Y - Yttrium	1/2
91Zr - Zirconium	5/2	93Nb - Niobium	9/2
95Mo - Molybdenum	5/2	97Mo - Molybdenum	5/2
99Tc - Technetium	9/2	99Ru - Ruthenium	5/2

101Ru - Ruthenium	5/2	103Rh - Rhodium	1/2
105Pd - Palladium	5/2	107Ag - Silver	1/2
109Ag - Silver	1/2	111Cd - Cadmium	1/2
113Cd - Cadmium	1/2	113In - Indium	9/2
115In - Indium	9/2	117Sn - Tin	1/2
119Sn - Tin	1/2	121Sb - Antimony	5/2
123Sb - Antimony	7/2	123Te - Tellurium	1/2
125Te - Tellurium	1/2	127I - Iodine	5/2
129Xe - Xenon	1/2	131Xe - Xenon	3/2
133Cs - Cesium	7/2	135Ba - Barium	3/2
137Ba - Barium	3/2	138La - Lanthanum	5
139La - Lanthanum	7/2	141Pr - Praseodymium	5/2
143Nd - Neodymium	7/2	145Nd - Neodymium	7/2
147Sm - Samarium	7/2	149Sm - Samarium	7/2
151Eu - Europium	5/2	153Eu - Europium	5/2
155Gd - Gadolinium	3/2	157Gd - Gadolinium	3/2
159Tb - Terbium	3/2	161Dy - Dysprosium	5/2
163Dy - Dysprosium	5/2	165Ho - Holmium	7/2
167Er - Erbium	7/2	169Tm - Thulium	1/2
171Yb - Ytterbium	1/2	173Yb - Ytterbium	5/2
175Lu - Lutetium	7/2	176Lu - Lutetium	7
177Hf - Hafnium	7/2	179Hf - Hafnium	9/2
181Ta - Tantalum	7/2	183W - Tungsten	1/2
185Re - Rhenium	5/2	187Re - Rhenium	5/2
187Os - Osmium	1/2	189Os - Osmium	3/2
191Ir - Iridium	3/2	193Ir - Iridium	3/2
195Pt - Platinum	1/2	197Au - Gold	3/2
199Hg - Mercury	1/2	201Hg - Mercury	3/2
203Tl - Thallium	1/2	205Tl - Thallium	1/2
207Pb - Lead	1/2	209Bi - Bismuth	9/2
235U - Uranium	7/2		

Information concerning available isotopes is obtained from the table in the ASCII file *isotope.list*. If desired, *isotope.list* can be edited but its **file format must be maintained**. For example, additional isotopes can be added to this list with names and/or quantum values which need not correspond to existing species, i.e. one may give arbitrary names and spin properties to “isotopes”.

It is recommended that all of the existing data in the file be left intact, and **it is essential that one not change the proton data**. Isotope labels should be restricted to 5 or less characters (*i.e.* 207Pb), element names should be no longer than 12 characters (*i.e.* Praeseodymium), and spin angular momentum values should be 3 or less characters (*i.e.* 7/2).

Gyromagnetic ratios of all isotopes are computed from their relative resonance frequencies (also found in *isotope.list*) based on the proton frequency by the formula

$$\gamma_i = \gamma_H \left[\frac{\Omega_i}{\Omega_H} \right], \quad (15-2)$$

where γ 's are the gyromagnetic ratios, and Ω the relative frequencies. Thus the gamma value of any isotope can be specified by setting its relative frequency in the table. Receptivity data in the table is currently not used. If, for some odd reason, multiple isotopes with the same designation are placed in *isotope.list* with the same name (e.g. two 23Na's), the one placed earliest in the list will be used by Isotope.